

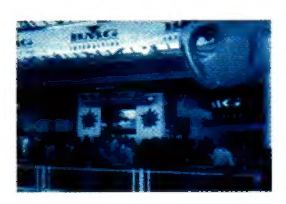
Doy la bienvenida a todos los apasionados del videojuego. GAME DEVELOPER ha sido desarrollado por los profesionales más cualificados del panorama nacional para iluminar vuestras mentes ansiosas de conocimiento y guiaros por los intrincados caminos del desarrollo de videojuegos.
Para aquellos más versados en el tema, garantizo que nunca se sabe demasiado de nada y que siempre se puede saber más de todo. Así que si queréis alcanzar un estado de evolución superior a los que os rodean, leer con detenimiento GAME DEVELOPER y descubriréis lo que es bueno.
Daremos respuesta a todas vuestras preguntas: programación, grafismo, infografía, composición musical, efectos de sonido, entrevistas con profesionales del sector, ..., no nos hemos olvidado de nada. Incluso se incluyen en nuestro CD de portada todos los ejemplos de la revista y el material necesario para que pongáis en práctica, desde el primer instante, todo lo que aquí contamos. Sin ninguna duda GAME DEVELOPER es lo que siempre habéis pedido y nadie hasta ahora os había sabido dar.



Premios ECTS 97

Los pasados días 7 al 9 de septiembre se ha celebrado en el Olimpia de Londres la entrega de los premios que, con carácter anual, se otorgan en la Feria Europea del videojuego a todos los expertos y profesionales del mundo del PC. Algunos de los galardones concedidos fueron los siguientes:

- **Mejor Hardware para PC:** 3Dfx Voodoo Graphics 3D Accelerator Chipset
- **Mejor marketing**
- **Oro:** Tomb Raider - EIDOS
- **Plata:** Platinum Range - Sony Computer Entertainment
- **Bronce:** C&C: Red Alert - Virgin
- **Desarrollador del Año:** Core Design
- **Juego PC del Año:** Tomb Raider - EIDOS
- **Distribuidor del Año:** EIDOS





Virgin refuerza su centro de poder

Brett W. Sperry, el actual presidente de Westwood Studios, compañía perteneciente a Virgin Interactive Entertainment, ha sido nombrado Presidente de Desarrollo Mundial por Martin Alper, presidente de VIE. B.W. Sperry ha sido, desde su fundación, la cabeza visible que ha estado tras algunos de los juegos más exitosos de los últimos años, entre ellos, por supuesto, la estrella de Westwood, la serie Command&Conquer, líder absoluto de superventas con más de 4 millones de unidades vendidas desde su aparición en el mercado, allá por el año 1995. Westwood Studios fue fundada por él junto a su socio Louis Castle hace doce años, en 1985, y VIE la adquirió en 1992. Durante los últimos doce años ha desarrollado joyas de la talla de Eye of the Beholder, Dune 2, Land of Lore o Kyrandia. Bajo la dirección de señor Sperry, VIE ha anunciado que actualmente tiene 25 títulos para Pc, así como numerosos proyectos en marcha para presentar durante el próximo año.

Sumario

The question

¿Por qué me tengo que comprar una aceleradora 3D?

Simple y llanamente porque es necesario para continuar caminando. La respuesta es la misma que dedujo el hombre al cuestionarse la necesidad de usar la rueda o seguir arrastrando las piedras por la fuerza. Las aceleradoras 3D simbolizan la evolución del hardware, el comienzo de una nueva era, igual que ocurrió con los discos de vinilo y el CD, o Windows 3.1 y el posterior Windows 95. Estas tarjetas 3D son buenas, son rápidas, son cómodas y, lo fundamental, es que no salen demasiado caras para el rendimiento que garantizan. Por alrededor de unas 30.000 pesetas puedes convertir tu Pc en una auténtica máquina infernal del videojuego. Así que ya sabéis, si queréis seguir en la brecha ir pensando en adquirir una de estas milagrosas tarjetas porque si la cosa sigue así, a finales del 98 la mayor parte de los lanzamientos punteros sólo funcionarán con aceleración hardware.
En el próximo número responderemos a la pregunta, ¿qué aceleradora 3d me compro?

Viejos conocidos abandonan el barco

De un tiempo a esta parte, numerosas figuras del mundo del videojuego, que hace unos años formaron sus propias desarrolladoras, han comenzado a abandonar el barco en el que navegaban con rumbo firme y seguro hacia el éxito. Dos claros ejemplos de esta supuestamente extraña tendencia son, sin ir más lejos, Peter Molineaux, ex-Presidente de Bullfrog y ex-Vicerpresidente de EA, al que debemos juegos de la talla de Populous, Theme Park o el reciente Dungeon Keeper, y John Romero, diseñador de Doom y Quake y ex-oráculo de Id Software. Ante esta desbandada merece la pena detenerse unos instantes a analizar la situación que ha desencadenado toda esta problemática. El factor común que ha envuelto todos estos casos de migración es claramente uno: pequeños grupos de desarrollo que, al ser absorbidos por una multinacional o alcanzar volúmenes de venta insospechados, cambian por completo su forma de concebir y desarrollar videojuegos. ¡La industria machaca al artesano una vez más! Esperemos que los genios sigan siendo genios aunque tengan a un comercial dirigiendo sus destinos.

Destacamos

El CD de portada incluye versiones de evaluación de las siguientes aplicaciones:

- **VISTA PRO:** generador de mundos virtuales en 3D.
- **COOL EDIT:** editor de samples para generar efectos de sonido de calidad.

Y además... las fuentes de código de los ejemplos comentados en nuestras secciones.

Autor: Fernando Benito

La programación de videojuegos

Muchas veces os habréis preguntado qué es lo que hay detrás de un vídeojuego. Y, a lo mejor, alguien os ha respondido y os ha hablado de matemáticas, zoomeado de sprites, motores 2D, motor 3D, luces, sombras, etc. Pero por encima de eso hay algo más: está el juego en sí mismo, lo que nos divierte, y que contaremos en esta serie de artículos.

Un médico, un ingeniero civil y una informática discutían acerca de cuál era la profesión más antigua del mundo. El médico señaló: "Bueno, en la Biblia se dice que Dios creó a Eva de una costilla que le quitó a Adán. Evidentemente esto requirió cirugía y por eso bien puedo afirmar que la mía es la profesión más antigua del mundo". El ingeniero interrumpió y dijo: "Pero incluso antes, en el Génesis, se dice que Dios creó el orden de los cielos y la tierra a partir del caos. Ésta fue la primera y, desde luego, la más espectacular aplicación de la ingeniería civil. Por tanto, querido doctor, está usted equivocado: la mía es la más antigua profesión del mundo". La informática se reclinó en la silla, sonrió y dijo tranquilamente: "Pero bueno, ¿quién pensáis que creó el caos?"
Con esta pequeña narración, los que todavía no halláis sufrido las inclemencias de algún compilador, podéis haceros una idea de la complejidad inherente a la creación de software.
Algunos de vosotros, que ya habréis hecho pinitos en la creación de videojuegos, pensaréis que tampoco es para tanto y que, realmente, lo único que hay que hacer para desarrollar esta labor es echarle ganas y tener algunos conocimientos de matemáticas y programación. Esto podía ser cierto en los tiempos de ordenadores como el ZX Spectrum, Comodore 64, Amstrad, etc. (allá por los entrañables años ochenta), pero llegados al punto de sofisticación que han alcanzado estos programas (comparemos las 48K de juegos de Spectrum como el *Knight Lore* con el CD-Rom 665.600K que ahora ocupa cualquier juego), ya no es suficiente con estas capacidades. No obstante, si todavía alguien no está de acuerdo, esperamos convencerle con estas páginas y, de paso, lograr que domine un poco más el difícil "arte" de programar videojuegos.
Uno de las avances más importantes que se han producido en este mundillo, en cuestión de ayudas a la programación de juegos, es la invención de la tecnología orientada a objetos, en definitiva: el modelo de objetos. Y uno de los lenguajes que la desarrolla de manera más o menos digna es el C++. Es importante reseñar que con esta serie de capítulos no se pretende enseñar a programar en C++, ya que se da por supuesto que se sabe (al menos C), por lo que sólo se verán algunos aspectos, seguramente los menos conocidos. Sin embargo, se intentará en todo momento comentar los fragmentos de código que se incluyan.

PROBLEMÁTICAS EN LA PROGRAMACIÓN DE VIDEOJUEGOS

En general, el software es complejo de forma innata. Un programa que plasme un videojuego no sólo consiste en un motor gráfico (cuestión de algoritmia pura), sino que hay algo más que, en la mayoría de los casos, es mucho más complejo y es lo que realmente hace el juego; de esto es de lo que se debería encargar la ingeniería del software.
Básicamente, esta complejidad se observa en cuatro elementos:

• **La complejidad del dominio del problema.** En el dominio de los grandes problemas a resolver existe una enorme cantidad de requisitos. Aplicado a los juegos existen unos requisitos funcionales como son la resolución en pantalla, la forma de impresión de los polígonos o *sprites*, la dinámica del juego (lo que debe hacer), etc. También hay otra serie de requerimientos que no son funcionales, como la facilidad de uso, el rendimiento (*framerate*), el coste, la fiabilidad, etc.
Otro problema bastante habitual es que los requisitos cambian durante el desarrollo. Esto es especialmente grave en el campo de los videojuegos, ya que como no suele existir una planificación demasiado buena, cualquier cambio resulta todo un mundo.
A veces es necesario que el software sea mantenible, es decir, si queremos que el software evolucione a lo largo del tiempo, éste tiene que ser fácil de cambiar para introducirle nuevas prestaciones o modificar otras que no eran óptimas. En los juegos este problema no suele ser muy problemático, ya que el juego no evoluciona demasiado, sino que es un ente que *se vende y se olvida*. De todas formas existen algunos casos de mantenimiento, como *Flight Simulator x*, *Crusader: no remorse/regret*, *Alone in the Dark x*, etc.

• **La dificultad de gestionar el proceso de desarrollo.** El tamaño no es una virtud para un sistema de software. Se intenta disminuir la cantidad de código, pero aún así a veces resulta complicado e imposible. En los juegos de cierta complejidad, donde hay involucrados más de dos programadores que no sean sólo de apoyo, nadie puede comprender completamente todo el sistema a título individual. Estamos hablando de decenas de miles de líneas de código, por lo que aún separando el código en módulos, se puede estar trabajando con cientos de ellos. Desde el punto de vista de la ingeniería del software todo esto constituye un reto, aunque no sea demasiado grande el equipo de desarrollo.
También hay que tener en cuenta que en la creación de un juego no sólo hay programadores, sino que participan grafistas, músicos y creativos (esos que se inventan las historias). Con estas personas también hay que *sufrir* una gran interacción sobre cómo han ser los datos a introducir en el programa.

• **La flexibilidad a alcanzar a través del software.** Una compañía de construcción de edificios no gestiona su propia explotación forestal para conseguir madera, ni construye una acería en la obra para suministrarse las vigas a medida. Sin embargo, en la industria del software de entretenimiento esto ocurre muy habitualmente. La flexibilidad que ofrece el software al desarrollador empuja a éste a construirse todos los bloques fundamentales

en los que se apoya el programa. Mientras que la industria de la construcción tiene normativas y estándares existe muy poco parecido en la industria del videojuego, con lo cual estos desarrollos suelen resultar sumamente lentos y laboriosos.

• **Problemas de caracterizar el comportamiento de sistemas discretos.** Si se lanza una pelota al aire, se puede prever el comportamiento que va a tener, aplicando ciertas leyes físicas, por lo que sería muy sorprendente si a mitad de vuelo saliera disparada hacia arriba. Este sistema es considerado como un sistema continuo basado en ecuaciones analógicas.
En el caso de los sistemas digitales, como los ordenadores, encontramos miles de variables que hacen que el número de flujos de control sea tan grande que resulte inabarcable el enumerarlos. En un videojuego, la cantidad de estados distintos en que puede encontrarse es enorme. Casi se podría asegurar que un programa de un juego no está lo suficientemente probado como para asegurar que no contiene errores, ya que no habría ni gente ni tiempo en el mundo para realizar todas las pruebas necesarias.
Hay tres aspectos fundamentales en los que habría que hacer especial hincapié y es que los juegos tienen que ser: **rápidos** (alto *framerate*), **atractivos** y, sobre todo, **divertidos**. La última de las características es la base de todo; si un juego no es divertido da igual todo lo demás.
Una vez vistos todos estos problemas que no son ni mucho menos insalvables, vamos a ver qué alternativas se nos ofrecen.

METODOLOGÍA ORIENTADA A OBJETOS

Ahora llega la gran pregunta: ¿qué es la programación orientada a objetos (POO)? Pues bien, si nos ponemos un tanto académicos una definición válida podría ser: *"La programación orientada a objetos es un método de desarrollo (implementación, como dicen algunos) en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia"*. (!Un ladrillo!).
Para los lectores poco habituados a este tipo de cosas, vamos aclarar la definición. Hay que destacar tres aspectos importantes de ella: la programación orientada a objetos utiliza **objetos**, no algoritmos, como sus bloques lógicos de construcción fundamentales; cada objeto es una **instancia** de alguna **clase**, y las clases están relacionadas con otras clases por medio de relaciones de **herencia**. Un programa puede parecer orientado a objetos, pero si falta cualquiera de estos tres elementos, ya no es un programa orientado a objetos.
Un objeto es una abstracción en la que se unen sentencias y datos, de tal forma que a un objeto sólo lo pueden tratar los métodos (funciones si se prefiere) definidos para él, y dichos métodos están específicamente preparados para trabajar con esa clase de objetos. Este grado de compenetración evita que un método pueda tratar datos no apropiados, o viceversa.
La clase es la piedra angular de la POO, ya que en torno a ella se crea todo y desde ella se puede obtener todo. Es como la creación de un nuevo tipo de dato, con la diferencia que está compuesta por datos y métodos. Se puede ver un ejemplo de la declaración de una clase en C++ en la figura 1 observándose todas estas cuestiones.
La herencia es otro de los pilares de la POO, ya que gracias a ella podemos reutilizar el código; asimismo, se pueden diseñar nuevas clases de objetos tomando como base otras ya existentes, de tal forma que el nuevo tipo cuenta con todo aquello que tenía el original, datos y métodos, pudiendo ampliar todo ello.

FIGURA 1. EJEMPLO DE CLASE OBJETO3D

```
class OBJETO3D
{
        protected:

        OBJETO3D **hijos;

        //BASICAS
        polar giro;
        vector posicion;
        vector pivote;
        vector escala;
        int pivotefijo;

        public:
        OBJETO3D();
        virtual ~OBJETO3D();

        //Escala el objeto 3D con los valores dados con respecto al pivote.
        void Escalar(vector v);

        //Mueve el objeto 3D en el espacio según el vector dado.
        void Mover(vector v);

        //Situa el objeto 3D en un lugar determinado del espacio.
        void Situar(vector v);

        //Desplaza el objeto 3D según un ángulo y un módulo.
        void Desplazar(polar p, double modulo);

        //Rota el objeto 3D en los ejes según los valores dados por el polar.
        void Rotar(polar p);

        //Pinta el objeto 3D según el tipo que sea.
        virtual void Pintar(DISPOS &d)=0;

        //Indica si el objeto 3D se ve o no.
        virtual int SeVe(DISPOS &d);
        ....................................
        ....................................
        ....................................
        ....................................
};
```

FIGURA 2. EJEMPLO DE CLASE DERIVADA CAMARA

```
class CAMARA : public OBJETO3D
{
        protected:

        //Variables relacionadas con la definición del objetivo.
        int ancho_objetivo;
        int alto_objetivo;
        int milimetros;

        public:
        CAMARA();
        ~CAMARA();

        ////////////////////////Métodos de la cámara.
        //Rota la cámara en los ejes según los valores dados por el polar.
        void Rotar(polar p);

        void Pintar(DISPOS &dis);

        //Mete en la cámara unos nuevos valores para el objetivo.
        void CambiarObjetivo(int ancho_objetivo, int alto_objetivo, int milimetros);
};
```

En la figura 2 se aprecia cómo la clase CAMARA deriva de la clase OBJETO3D, teniendo los mismos datos y pudiendo realizar las mismas cosas que podía ésta; pero, además, tiene valores como *ancho_objetivo* que no existía en OBJETO3D y métodos como CambiarObjetivo también nuevo.
Ahora nos detendremos en los problemas que presenta la programación de videojuegos y, posteriormente, en las soluciones que ofrece a algunos de estos problemas la POO.

SOLUCIONES QUE NOS DA LA POO

Éstas son las que nos aportan sus cuatro elementos fundamentales: la abstracción, el encapsulamiento, la modularidad y la jerarquía.
La **abstracción** se centra en las características fundamentales que un observador percibe de algún objeto y es una de las vías fundamentales por las que los humanos combatimos la complejidad. Una abstracción es una visión simplificada de un sistema que remarca algunas propiedades del mismo y suprime otras. Una buena abstracción es aquella que enfatiza detalles significativos, suprimiendo los irrelevantes.
La manera en que el C++ desarrolla esta cualidad es la clase. A través de ella se expresan los rasgos de los objetos.
El **encapsulamiento** y la abstracción son conceptos complementarios: la abstracción se centra en el comportamiento observable de un objeto mientras que el encapsulamiento se encarga de la implementación que da lugar a este comportamiento. Con este proceso se trata de ocultar la información, de que esté oculta la estructura interna del objeto, así como la implementación de sus métodos.
De esta forma se nos provee de un interfaz a través del cuál interactuamos con el objeto sin saber lo que hay dentro de él, por lo que cualquier programa construido a base de objetos sea más fácil de mantener, ya que si queremos cambiar algo, sólo tendremos que cambiar su implementación, sin tener que "molestar" al resto de componentes cambiando el interfaz.
La forma en que el C++ nos proporciona este encapsulamiento es haciendo privadas, por defecto, todas las variables y métodos que hay dentro de una clase. Así sólo los métodos de la propia clase pueden hacer uso de las variables de la misma. Existen maneras de hacer públicos tanto variables como métodos, mediante las palabras clave *public* y *protected*.
Modularidad es el acto de fragmentar un programa en componentes individuales, que puede reducir su complejidad en algún grado. Realmente la división en módulos crea unas fronteras que ayuda a la mejor comprensión del programa. La decisión sobre el conjunto adecuado de módulos es muy importante: una buena separación traerá una mayor facilidad en el desarrollo.
Un buen módulo se caracteriza por tener gran cohesión interna, es decir, que todo lo que se haga dentro de él esté íntimamente relacionado, y un débil acoplamiento con el resto de módulos, o sea, que la relación con otros módulos no sea demasiado fuerte.
Los módulos en C++ no son más que ficheros compilados separadamente. Se suele hacer uso de un fichero de cabecera ".hpp", donde se colocan las interfaces de los módulos y de un fichero de implementación con extensión ".cpp".
Por su parte, la **jerarquía** consiste en una clasificación u ordenación de abstracciones. Asimismo, se puede expresar como una manera de organizar las clases, de relacionarlas entre sí cuando el problema empieza a ser complejo.
Esto se expresa en C++ a través de la herencia que puede ser simple cuando la clase hija deriva de una sola clase, o múltiple cuando se deriva de más de una clase. No es muy recomendable el uso de la herencia múltiple, ya que introduce una gran complejidad adicional.
Cabe reseñar también como soluciones positivas de la POO la creación de objetos en tiempo de ejecución, muy apropiado para los juegos, en que hay elementos que se crean de la nada como explosiones, disparos y otras cosas que aparecen. Esto se hace en C++ mediante *new*. Al igual que los objetos se crean, también se destruyen con *delete*.
La concurrencia también es un aspecto a tener en cuenta en la POO. Los objetos tienen tal entidad por sí mismos que pueden actuar en solitario sin seguir ningún tipo de secuencia preestablecida, es decir, pueden actuar a la vez. Esta cuestión no está muy explotada, ya que los sistemas multiprocesador no están demasiado difundidos a nivel de usuario de videojuegos; además, los sistemas de reparto de carga computacional en sistemas multiprocesador si bien no están demasiado avanzados tienen gran futuro. Imaginaos lo que sucedería si a cada objeto creado se le asignara un procesador y que entre ellos se mandaran mensajes (¡PURA DINAMITA!) y actuaran en colaboración, incluso en disputa unos con otros por recursos de servidores. (Pensad en las posibilidades y la velocidad del invento. !Es de CIENCIA FICCIÓN!)
En general, los programas realizados con un diseño y un lenguaje orientados a objetos son más reutilizables, más pequeños, más flexibles al cambio, aptos para reducir los riesgos de desarrollos complejos y más fáciles de entender para la mente humana, que tiene bastante asumida la idea de objeto.
Por ello, en el siguiente capítulo se hará una introducción al C++ para comprobar lo intuitivo que resulta convertir objetos clásicos de un videojuego en abstracciones en C++ entendibles por el ordenador.



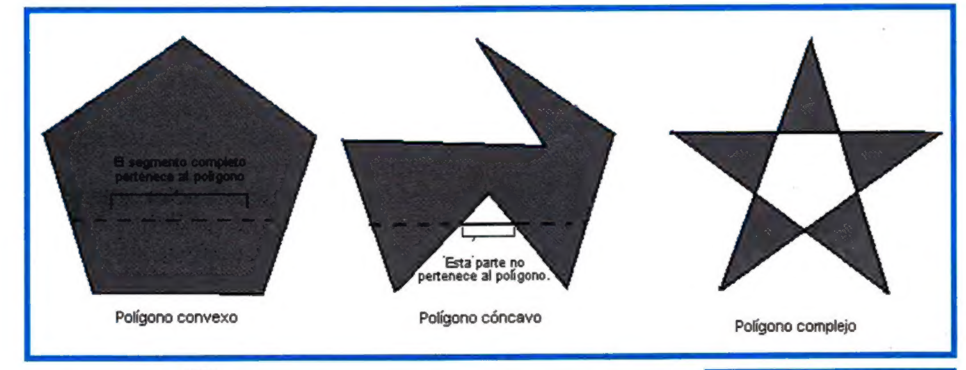
Autor: Alberto García-Baquero

Polígonos

Actualmente, las tres dimensiones están tomando un papel cada vez más importante en los videojuegos. La mayoría de los lanzamientos importantes de las compañías de software intentan conseguir un look tridimensional. El soporte para las recientes tarjetas 3D es requisito indispensable.

Ante esta clara revolución de las 3D cualquier programador que quiera desarrollar un videojuego con perspectivas serias de mercado, y no desee quedarse atrás tecnológicamente hablando, no le resta otra salida que meterse de lleno en las 3D. Por este motivo, con idea de ayudar a los que están aprendiendo y completar los conocimientos de los entendidos comenzamos esta sección dedicada por completo a todo lo concerniente a la programación de 3D enfocada al desarrollo de videojuegos, donde esperamos encontréis un buen lugar de referencia en el que añadir a los conocimientos básicos, información interesante, útil y original, ya que hay mucho escrito sobre este tema.
Para empezar con buen pie, no tenemos más remedio que empezar con los polígonos: qué son (*definición*), cuántos tipos de polígonos hay (*clasificación*) y cómo se dibujan en la pantalla (*representación*).
A pesar de existir múltiples técnicas para la representación de entornos/objetos tridimensionales, tales como *voxel*, *raycasting*, *raytracing*, *NURBS*, etc., la representación de modelos poligonales es el método más utilizado en el mundo de los videojuegos, ya que puede conseguirse con ellos un buen *frame rate* y están exentos de las limitaciones del voxel (Comanche, por ejemplo) y del *raycasting* (Doom, por ejemplo) pudiéndose conseguir hasta 6 DOF (grados de libertad de visión) en tiempo real y permitiéndose la construcción de mundos realmente complejos utilizando la gran cantidad de herramientas de excelente calidad existentes en el mercado, tales como 3DS MAX, LightWave, Alias Power Animator, etc. Todo esto, junto con el apoyo de las tarjetas 3D que liberan al procesador del dibujado de polígonos, da como resultado que el polígono será la base de la representación de objetos tridimensionales de aquí a un buen tiempo.
Podemos definir un polígono como un objeto 2D contenido en un único plano con un área delimitada por un número finito de lados.
Los polígonos pueden clasificarse en varios tipos según sus diferentes cualidades. Nosotros nos limitaremos a clasificarlos según su forma; así éstos pueden ser convexos, cóncavos o complejos. Un polígono convexo es aquel en el que el ángulo interior de cada uno de sus vértices es menor o igual que 180 grados, en caso contrario este es cóncavo. Otra definición de polígonos convexos sería que cualquier segmento que una dos puntos de un polígono convexo tiene todos sus puntos pertenecientes al polígono. Los polígonos complejos tienen sus lados intersectantes entre sí, con lo que pueden llegar a determinar varias áreas o áreas con agujeros como los ejemplos que se ilustran en las figuras. En este artículo explicaremos cómo dibujar tanto los polígonos convexos como los cóncavos y complejos, que aunque suelen ser menos usados, ya que se pueden simplificar en polígonos convexos, pueden ser de gran ayuda en varios casos contribuyendo a disminuir radicalmente el número de polígonos pertenecientes a un mismo plano.
El método que utilizaremos para dibujar los polígonos es recorrerlos de arriba a abajo e ir dibujando cada sección horizontal correspondiente. Para saber si un punto se encuentra dentro o no de un polígono, se toma una sección cualquiera del polígono tomada desde el infinito hasta este punto; si el número de intersecciones con los lados del polígono es par, el punto está fuera del polígono y si es impar el punto se encuentra dentro del polígono. A esta regla se le llama la regla del impar-par (*even-odd rule*), según la cual por cada línea horizontal, para dibujar todos los puntos pertenecientes al polígono, debemos trazar una línea entre las intersecciones 1 y 2, 3 y 4, 5 y 6,..., n-1 y n.
En el caso de los polígonos convexos, esta sección horizontal del polígono tendrá tan solo dos intersecciones con los lados de éste que serán los puntos extremos del mismo en esa sección ya que como anteriormente se comentó, todos los puntos de un segmento que unan dos puntos pertenecientes a un polígono convexo también pertenecen a este. Así, para dibujar un polígono convexo sólo tenemos que recorrer de arriba a abajo el polígono, y por cada línea horizontal dibujar una línea desde el extremo izquierdo al derecho de la sección horizontal, con el polígono.
En el caso de los polígonos cóncavos y complejos es posible que nos encontremos con más de un segmento por sección horizontal, ya que carecen de la propiedad de los polígonos convexos; por este motivo y tal como veremos más adelante será necesario mantener una lista de ejes activos (*Active Edge Table*, AET) para poder manejar fácilmente la múltiple cantidad de ejes que son seccionados por una misma línea horizontal.
Sin embargo, aunque hasta ahora el proceso pueda parecer sencillo no lo es tanto, ya que para que no haya solapamiento entre polígonos vecinos es necesario determinar una norma en la que se defina qué puntos están 'dentro' del polígono y cuáles no, tarea más complicada de lo que pueda parecer a priori, para que

FIGURA 1.



LISTADO 1

```
//
// Primero calculamos la pendiente del lado o eje en forma
// entera + fracción.
//
Numerador   = (int)(NextP->x  - P->x);
Denominador  = (int)(NextP->y) - (int)(P->y);

XStep         = Numerador / Denominador;
if( Numerador > 0 )    Numerador   %= Denominador;
else
{
  //
  // Situación especial con los deltas negativos
  //
  Numerador  = (-Numerador) % Denominador;
  if( Numerador )
    {
      XStep—;
      Numerador = Denominador - Numerador;
    }
}
// Ponemos los correspondientes valores inciales
X       = P->x;
XError = Numerador;
```

así ningún polígono solape (ni por un solo punto) ninguno de sus polígonos adyacentes. Las normas que tomaremos en este artículo serán las siguientes:

- Los puntos situados exactamente sobre ejes no horizontales son dibujados solamente si el interior del polígono está a la derecha (los ejes de la izquierda se pintan, los de la derecha no).
- Los puntos situados exactamente sobre los ejes horizontales son únicamente dibujados si el interior del polígono está bajo ellos (ejes horizontales superiores se pintan, los inferiores no).
- Un vértice es dibujado si todas las líneas que terminan en este punto cumplen las anteriores condiciones (ningún eje derecho o inferior termina en este punto).

Estas reglas podrían haber sido tomadas en cualquier otro sentido; el propósito es evitar el solapamiento de los polígonos adyacentes, pudiendo valer distintas configuraciones para el mismo fin.

Con toda esta teoría ya presentada llegó el momento de ponerse manos a la obra.

Empezaremos con los polígonos convexos ya que son más sencillos de dibujar; un polígono viene determinado por una lista ordenada de puntos en el plano, que en el caso de los polígonos convexos puede venir dada en sentido horario o antihorario que, si bien no es imprescindible a la hora del dibujado del polígono en sí, sirve de ayuda para la optimización y simplificación del algoritmo y, además, será de gran utilidad e importancia cuando procedamos más adelante a eliminar las caras ocultas.

El primer paso será encontrar los vértices superior e inferior del polígono. A partir del vértice superior, calcularemos la pendiente del lado que va desde este vértice al próximo en la lista que, en caso de haber tomado la norma de dar los polígonos en forma horaria, será el eje del extremo derecho; deberemos calcular también la pendiente para el lado del extremo izquierdo, que partirá desde el vértice superior al vértice inmediatamente anterior en la lista o, en su defecto, el último de la lista, ya que aunque importe el sentido en que vengan ordenados los puntos, al ir conectados el primero de la lista con el último, y viceversa, no es similar por cuál se tome como inicio. En el caso de que algún eje sea horizontal, éste se descarta y se pasa a procesar el siguiente.

A partir de la pendiente calculada por cada eje podemos saber la intersección con el eje por cada sección horizontal sumándole la pendiente al valor anterior. Esta pendiente puede ser calculada en números de coma flotante, en números de coma-fija (*fixed-point*) o mediante un DDA (*digital differential analyzer*). El uso de flotantes queda prácticamente descartado ya que, aún a pesar de que es cada vez más asequible gracias a los modernos procesadores, en este caso es innecesario ya que los números en coma-fija y DDA hacen mucho mejor y más rápido este papel. Los números en coma fija son números enteros pero multiplicados por una constante determinada para guardar más precisión siendo su valor real en flotante el mismo número dividido por la constante; ésta suele ser una potencia de dos, simplificando las operaciones a simples desplazamientos de bits. Cuanto mayor numero de *bits* se tome para la parte no entera, mayor precisión se guardará. Este método es muy utilizado por mucha gente, ya que es rápido y facilita la optimización de código. Aquí utilizaremos el método por DDA, que contempla todas las ventajas de la coma fija y aporta algunas más que nos serán de gran ayuda más adelante como es, por ejemplo, tener constantemente los valores en formato de enteros puros sin necesidad de conversiones continuas, controlar directamente el acarreo de la parte no entera, siendo incluso notablemente más preciso este método que el uso de coma flotante (1/3 es más preciso que 0.3 periódico por ejemplo), pudiendo actuar diferentemente según haya o no acarreo e incluso ahorrándonos la división correspondiente de la pendiente cuando ésta sea menor que 1, caso que suele ocurrir el 50% de las veces.

El método de DDA consiste en guardar los números en forma de fracción, así 4.23 se puede representar como 423/100 o mejor aún 4+ 23/100. Para calcular la posición de las secciones del eje correspondiente por cada línea horizontal, lo haremos a partir del valor de la línea anterior, así:

$$x_{i+1} = x_i + \frac{1}{m} \text{ , siendo } m = \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

la pendiente de la línea.

Este incremento tendrá como resultado que *x* tenga una parte entera y una parte fraccional, que puede ser expresada mediante una fracción de denominador

$$y_{max} - y_{min}.$$

Conforme avanzamos en este proceso, la parte fraccional sobrepasará el valor del denominador y la parte entera deberá ser incrementada. Así, por ejemplo, si la pendiente fuera 1/3 y x_{min} valiera 2, la secuencia de los valores de x sería, que sería

$$2, 2\frac{1}{3}, 2\frac{2}{3}, 2\frac{3}{3}$$

LISTADO 2

```
X              += XStep;
  XError += Numerador;
  if( XError >= Denominador )
    {
    X++;
    XError -= Denominador;
    }
```

igual a 3, y así hasta llegar a la y_{max}.
En el Listado 1 podemos ver el proceso de inicialización de un eje; primero calculamos el numerador y el denominador de la pendiente. Posteriormente, en caso de que la pendiente sea mayor que uno, extraemos la parte entera y dejamos al cociente con la parte decimal. Aunque en el código de ejemplo la división se lleva a cabo siempre, ésta es tan solo necesaria cuando la pendiente sea mayor que 1, consiguiendo con ello una gran optimización, ya que la división es una de las operaciones que más tiempo consume, pudiendo ser evadida por este método, llegándose a dibujar polígonos enteros sin necesidad de una sola división. En el caso de que la pendiente sea negativa habrá que tratar las operaciones de forma especial tal como se muestra en el Listado 1. Tras calcular los valores de la pendiente debemos incializar los valores de inicio como la coordenada x inicial.
Una vez inicializados los ejes extremos izquierdo y derecho, pintamos la línea correspondiente y, posteriormente, actualizamos los ejes para el avance de línea tal y como muestra el Listado 2. Primero, sumamos al valor anterior la parte entera de la pendiente y después a la parte fraccional donde se va registrando el error acumulado, se le suma la parte fraccional de la pendiente, el numerador. Posteriormente comprobamos si el error acumulado es mayor que 1, en cuyo caso procederemos a sumarle una unidad a la parte entera y restársela a la parte fraccional, manteniendo siempre el error acumulado por debajo de 1. Así, seguimos rastreando líneas hasta que lleguemos a un cambio de eje con su correspondiente inicialización, que en caso de ser uno horizontal se pasará al siguiente, o hasta que se llegue a la línea inmediatamente anterior a donde está situado el vértice inferior. Una vez llegados a este punto el polígono ya está dibujado al completo.
Para los polígonos cóncavos y complejos el procedimiento a seguir es algo distinto ya que puede darse el caso de que existan más de dos extremos por sección horizontal. Para empezar, calcularemos la lista global de ejes, donde a excepción de los ejes horizontales, incluiremos la información de todos los ejes válidos ya inicializados. Para tener un mayor control de las líneas de cambio e inicio de eje, a medida que vamos añadiendo ejes a la lista global, insertaremos la información de cuándo comienza y termina cada línea, en una lista ordenada que llamaremos de 'eventos'; este procedimiento a pesar de no ser el único posible si es el más claro y simple y, por lo tanto, rápido ya que nos ahorrará por cada línea el tener que comprobar eje por eje si comienza a estar activo o no. Tras este paso e inicializar los valores correspondientes, sólo nos queda empezar el proceso de rastreo de líneas, que seguirá los siguientes pasos:

1. Controlar los cambios de segmento, activando los ejes de la lista global que comiencen a dibujarse en la línea actual y desactivando los que finalicen en esta línea. En este proceso es donde utilizaremos la lista de 'eventos', de este modo las comprobaciones de todos los ejes se reducen a una simple comparación por línea.
2. La lista de ejes activos se ordena de izquierda a derecha.
3. Actualizamos los ejes activos para calcular los valores en la siguiente línea.
4. Pintamos las líneas correspondientes según la regla del impar-par *(even-odd rule)*
5. Se repite todo el proceso hasta la línea inferior.

Para los polígonos cóncavos podrían añadirse varias optimizaciones, ya que el orden de los ejes es constante para todo el polígono, puesto que los ejes no se cruzan entre sí como pasa en los polígonos complejos. Así pues, la lista de ejes debería ordenarse de izquierda a derecha en el proceso de inicialización antes de empezar el rastreo. De esta manera, tomamos los valores ordenados previamente y no necesitamos el paso 2. El implementar esta optimización sería un buen ejercicio para los lectores que estén interesados ya que no requiere muchas modificaciones a partir de la función de dibujar un polígono complejo.
En el próximo número profundizaremos en la interpolación a lo largo de un plano 3D proyectado en una superficie 2D, enfocándolo en sus aplicaciones más prácticas, textureado e iluminación *gouraud* de polígonos. Se explicarán los distintos métodos y aproximaciones posibles: interpolación lineal, cuadrática, hiperbólica, subdivisión, etc. También discutiremos sobre la conveniencia en el uso de triángulos o polígonos de n-lados, los pros y los contras de cada uno.

Alberto García-Baquero
wisefox@cyberdude.com
albertogb@jet.es
wisefox@jet.es

Game Developer al rescate

Colabora con GAME DEVELOPER y descubre de lo que eres capaz. Te revelamos las secciones en las que estamos trabajando duro y decide si quieres unir tu esfuerzo al nuestro para conseguir lo que sólo unos pocos han conseguido hasta el momento.

SECCIÓN GURU CODERS
Si tienes talento y controlas de programación, eres un demoero de pro o tu código es el más rápido de este lado del hemisferio, no te conformes con que sólo lo sepan tus amigos y demuestra al mundo de los que eres capaz. Envíanos muestras de tu trabajo junto con una carta de presentación en la que nos cuentes todo aquello que te interese que sepamos: desde cuando codeas, cuántas horas resistes frente al teclado, qué opinas de John Carmak, ... Prometemos publicar todo el material que recibamos (lo que venga después depende del destino). No que quedes atrás y descubre que tu también vales millones.

SECCIÓN MAX QUE AMIGOS
Si te ríes en la cara de las estaciones Silicon Graphics y defiendes que con un 486 a 66Mhz y 8Mb de RAM podrías hacer tu solo y tu 3D MAX la película *Toy Story*. Si no te hace falta un scanner 3D para reproducir al átomo la Torre de Pisa, no lo flipes más y mándanos imágenes que lo prueben. Convence al mundo de que el arte está por encima de la máquina que lo sustenta. Envíanos imágenes de tus render o manda tus animaciones (aconsejamos usar cartuchos ZIP) o quizá prefieras colocarnos un vídeo VHS, nos da lo mismo el soporte. Juzgaremos tu trabajo y te comentaremos nuestra opinión; es posible que exista un nuevo John Lasseter entre nosotros.

SECCIÓN MERCENARIOS DEL VIDEOJUEGO
Si sois un grupo de amigos que desarrolláis videojuegos por afición pero queréis empezar a hacerlo por dinero, en estas páginas os comprendemos y os ayudamos. Contarnos los problemas que os plantea el difícil proceso del desarrollo (falta de máquinas, de gente, de coordinación, de ideas, ...) y desde aquí os aconsejaremos lo mejor que podamos. Mandad vuestro trabajo y os asesoraremos sobre todo lo que queráis saber acerca del desarrollo de un videojuego.

Envía todo tu material (discos, ZIP, CD-ROM, Vídeos) a la dirección que te indicamos; te responderemos en un breve plazo. Por supuesto, contactaremos contigo para conocer más detalles sobre tu obra:

GAME OVER
c/ Alfonso Gómez, 42 NAVE 1-1-2,
28037 Madrid
(España)



CÓMO HACER TROMPOS EN C++

Autor: Jorge Rosado

Coders sobre ruedas

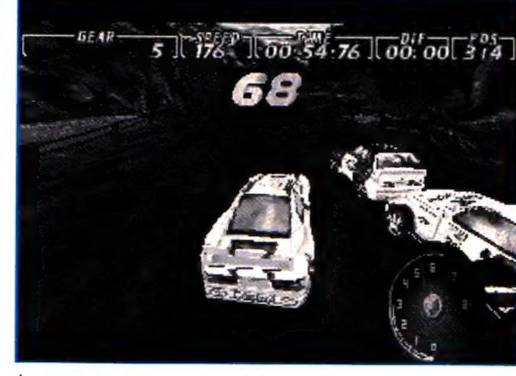
Aprovechando que nuestro piloto de moda Carlos Sainz se acerca cada vez más al campeonato de rallyes y siguiendo con la moda de arcades de carreras tipo Sega Rally, la compañía española Digital Dreams Multimedia nos presenta su aportación al mundo del motor; nosotros hablamos con sus creadores.

Un sugerente simulador automovilístico de nombre *World Wide Rally* llega a nuestro banco de pruebas. Después de contemplar una demo del juego nos quedamos tan alucinados con él, que decidimos encaminar nuestros pasos hasta el cuartel general del grupo DDM para realizar una entrevista a sus programadores. Una vez allí nos presentaron a dos curtidos muchachos, Javier y David, que nos recibieron con los brazos abiertos y se mostraron muy complacientes a la hora de dar respuesta a nuestras incisivas cuestiones. Pero, antes de pasar a la entrevista, os comentaremos las excelencias de World Wide Rally.

Lo primero que cabe destacar es que su engine 3D no tiene nada que envidiar a la de juegos tan conocidos como *The Need for Speed I y II* o *Screamer*, polígonos texturados, corrección de perspectiva, modelos 3D con animación, ..., es decir todo lo que un buen engine debe tener. Pero lo que verdaderamente destaca del resto de características en este juego es su motor físico.



DISFRUTAREMOS DE FANTASTICAS ACROBACIAS.



LA SENSACION DE VELOCIDAD ES INSUPERABLE.

Derrapes, vuelcos, choques, saltos, ..., pero no con animaciones precalculadas, nada de eso, todos los modelos siguen unas constantes físicas primarias como son el peso, la gravedad, el rozamiento o el empuje lo que provoca que, según la localización del choque, la carrocería y la orografía de la pista los coches realicen trompos, den vueltas de campana, ...

Otro dato a resaltar es que a pesar de que no posee la licencia oficial del mundial de rallyes, se han reproducido con detalle los coches más significativos que compiten en todos los equipos, Toyota, Subaru, Lancia, ..., incluso han incluido un pequeño jeep para romper la monotonía, consiguiendo, con esto, un elevado número de modelos que sumados a las variadas y detalladas pistas y a sus modos de competición hacen que las horas de diversión delante de World Wide Rally aumenten hasta niveles peligrosos (se me olvidaba algo: también podréis disfrutar de un divertido modo multijugador).

En definitiva un juego que no defrauda que podréis disfrutar de inmediato ya que este mes sale a la venta en todos los puntos de venta del territorio nacional al módico precio de 2995 pesetas, todo un lujo a un precio de risa. Desde aquí os recomendamos su compra. Saludos.













LO QUE MÁS DETESTAIS LOS PROGRAMADORES DE JUEGOS

- El sonido del Windows95 arrancando.
- Las sugerencias e "ideas" para el juego.
- Cambiar de compilador.
- El lanzamiento a mitad de proyecto de un juego de la competencia virtualmente idéntico.
- Los teclados especiales "WINDOWS 95".

1. ¿Desde cuándo lleváis programando videojuegos?
Llevamos aproximadamente dos años desarrollando videojuegos de forma profesional. Pero realmente empezamos a programar juegos a los 16 años, con el mítico Spectrum. Pertenecemos pues a toda esa generacion de programadores que aprendimos con esa pequeña maravilla de Sinclair.
Es curioso que ahora no exista un equivalente actual al Spectrum, puesto que ahora, para un principiante es casi imposible alcanzar el conocimiento total del funcionamiento de un PC. Sin embargo, conceptos tales como la pila, el buffer de vídeo o las interrupciones eran casi intuitivos en ese ordenador.

2. ¿Por qué decidistéis hacer un juego de rallyes?
La simulacion automovilística es un genero "clásico" dentro de los videojuegos para ordenador. Siempre me han parecido muy divertidos todos los juegos de este género.
El "empujón" definitivo fue la avalancha de máquinas recreativas que inundó los salones hará dos años. Después de gastarnos hasta el ultimo duro en ellas, decidimos intentar conseguir la calidad de las recreativas en la plataforma PC.

3. ¿Qué novedades incorpora vuestro juego World Wide Rally respecto otros juegos de carreras?
Básicamente velocidad. Pensamos que el mejor videojuego es aquél que refresca la pantalla más veces por segundo. Un "framerate" alto es imprescindible para dar al jugador la sensación de velocidad. También es importante la corrección de perspectiva en las texturas, con lo que se evita ese "alargamiento" extraño que aparece en la parte baja de la pantalla tan frecuentemente.

4. ¿Qué resultó más duro durante el desarrollo?
En principio la optimizacion del motor 3D. Fue realmente muy difícil el conseguir "arañar" ciclos de CPU en las tareas más críticas -como por ejemplo la rutina final de texturas-. También fue bastante compleja la edicion de las pistas, dado que a mitad de proyecto se cambió el formato de las mismas y tuvieron que empezarse de nuevo.
También conforme la fecha de finalización se iba acercando, se convirtió en tradición el programar por las noches subsistiendo a base de increíbles cantidades de café.

5. ¿Cuánto tiempo os llevo el desarrollo de World Wide Rally?
En lo que a programación de la parte arcade respecta, básicamente un año. Aunque hay otras tareas paralelas como la programación del instalador, setup, menús, y un largo etc. Además, siempre suelen surgir al final pequeños problemas que habían pasado inadvertidos, con lo que hay que reprogramar parte del juego.
En total, y contando todo esto, el juego tardó realmente un año y medio en realizarse.

6. ¿Facilita la programación trabajar entre dos o es preferible hacerlo todo solo?
Depende de cada uno. En nuestro caso consideramos positiva la experiencia, puesto que nos fuimos especializando en partes concretas del juego: Motor3D, física, opción multijugador...
Sin embargo, creemos que la mayoría de los programadores no estarán de acuerdo con eso.


DAVID PICON Y JAVIER CARRION, LOS CODERS DEL RALLY.

Puesto que cada uno tiene un "estilo" de programar diferente y por regla general no compatible con el de otros.

7. ¿En qué otros desarrollos estáis trabajando?
Ahora estamos empezando un ambicioso proyecto ambientado en la Segunda Guerra Mundial. Básicamente es un simulador de vuelo al estilo "*SECRET WEAPONS OF THE LUFTWAFFE*" pero con las técnicas 3D actuales. Aunque lo más destacable del juego es la capacidad de poder jugar partidas multijugador vía internet.
Creemos que el futuro de los juegos radica en la posibilidad de jugar contra otras personas y no contra una IA por muy bien diseñada que esté.

8. ¿Cómo creéis que afectará la llegada de las aceleradoras 3D a los futuros juegos 3D?
En principio resultará positiva. Es impresionante el aumento de velocidad que consiguen algunas tarjetas. El problema radica en la no estandarización de las mismas. Resulta desesperante el tener que programar drivers para cada una de las tarjetas 3D que hay en el mercado. De todas formas parece que estamos asistiendo a una "estandarización" por lo que es probable que en menos de un año sólo haya dos tipos principales de tarjetas (como sucedió en su día con la Sound Blaster). Todos sabemos cuáles son.

9. Exceptuando World Wide Rally, decidme vuestro juego de carrreras favorito.
En nuestra opinión, *The need for Speed I*. Tiene el modelo de simulación física más perfecto del mercado, las pistas son divertidas y los sonidos muy buenos. Lo único que no nos agrada es la limitación del ancho de las pistas y la escasa velocidad que daba en SVGA. Pero es, sin duda, un magnífico juego.

10. ¿Por qué los programadores de juegos odiáis Windows?
La idea de un entorno gráfico multitarea al estilo Macintosh para PC es maravillosa. Pudo ser buenísima, como el entorno X Windows (UNIX), pero la filosofía que Microsoft ha puesto en Windows 95 es la de nunca entregar el control total a la aplicación. Resulta inconcebible que alguien quiera jugar a un videojuego, mandar un fax e imprimir un informe todo a la vez. Pero para que pudiera hacerse, se pierde el control directo del teclado, CPU, memoria, etc..

LO QUE MÁS OS AGRADA A LOS PROGRAMADORES DE JUEGOS

- Ver cómo el juego de la competencia recién lanzado se bloquea.
- Compilar sin errores a la primera y que funcione.
- Enseñar a todo ser viviente la rutina que acabamos de programar.
- Un buen teclado mecánico.
- Ver el juego en las estanterías de los grandes almacenes y que se está vendiendo.

Autor: Víctor Segura

Música en los videojuegos

Atrás quedaron los Amiga, los Atari y demás ordenadores incompatibles entre sí destinados, entre otras cosas, la creación musical. Con la unificación de hardware y software de sonido en los PC's, la música sintética llama a las puertas de nuestra casa para hacerse un hueco entre nosotros.

Durante los próximos meses analizaremos los diferentes equipos de sonido (teclados, módulos, etc) y programas de música (secuenciadores, trackers, editores de ondas etc) que se utilizan en el desarrollo de videojuegos, con el fin de familiarizarnos con ellos lo mejor posible y saber, exactamente, qué debemos comprar si queremos hacer alguna actividad musical con la ayuda de los ordenadores, sin caer en el engaño de los precios y de las marcas, simplemente guiados por nuestros conocimientos y necesidades.

EN BUSCA DE UN FORMATO ESTANDAR

Al igual que ocurre en la mayoría de los campos informáticos, en lo referente a la música existen muchos formatos diferentes. Los más comunes son los formatos MOD, S3M, XM y MID, siendo éste último el que posee una mayor aceptación en el mercado "informático musical"; de hecho, los compositores profesionales, antes de llevar su música a una orquesta para que ésta la interprete, hacen los arreglos en sistemas que soporten este formato. (Cualquier PC equipado con una tarjeta de sonido de gama media ofrece la posibilidad de hacer "Música Midi").

MIDI GANA LA BATALLA

El MIDI (*Musical Instrument Digital Interface*) es como un "lenguaje musical" que, a diferencia de otros tipos de "lenguajes", se ha establecido (y no sólo eso, sino que también evoluciona trepidantemente) de manera que hoy en día, cualquier instrumento digital y cualquier tarjeta de sonido posee un interfaz midi para poder aprovechar todas las ventajas que este lenguaje nos proporciona.

Analizaremos todo tipo de software y hardware musical

Durante los meses que siguen estudiaremos, las mejores herramientas para poder hacer lo que a partir de ahora llamaremos "Música Midi" (que es la utilizada para hacer las canciones que componen la banda sonora de un videojuego).

La importancia de la música en el videojuego

Años atrás, la música era prácticamente prescindible en el desarrollo de un videojuego ya que daba igual que éste tuviera o no música. Afortunadamente esta idea ha cambiado mucho a lo largo de estos últimos años, de tal manera que las productoras, a la hora de realizar un videojuego destinan gran parte del dinero para el desarrollo musical. Y, además, a la hora de hablar de las canciones que forman un videojuego no sólo se habla de música, sino que, en ocasiones, se habla de "Banda Sonora".

Si analizamos el aspecto sonoro de un videojuego nos encontramos con dos puntos muy importantes: la música y los efectos de sonido. Tenemos que tener en cuenta que tanto las canciones que componen un videojuego, como los efectos de sonido son muy importantes en el desarrollo sonoro del videojuego, por lo tanto, hay que prestar especial atención a los programas para hacer música (secuenciadores y trackers) y efectos de sonido (editores de ondas), sin olvidar otro tipo de utilidades interesantes que no están relacionadas directamente con la creación de la música en los videojuegos pero que nos serán de gran ayuda.

Mesa de mezclas incluida en un secuenciador.

LOS EFECTOS DE SONIDO

Una vez acabada la banda sonora del viedeojuego nos enfrentamos a una tarea que, sin ser tan creativa como hacer música, es igualmente importante. Además, si poseemos un control absoluto en la elaboración de efectos de sonido, podremos crear cualquier tipo de sonido que podrá ser utilizado como instrumento para la elaboración de nuestra música. A diferencia del software musical, en el campo de los efectos de sonido los programas que existen son bastante similares entre sí. El formato más extendido es el WAV. Explicaremos cómo es este formato, cómo aplicar diferentes efectos sobre un sonido (eco, flanger, reberv, etc.) y todo lo que necesitamos saber para hacer unos buenos efectos de sonido, que pongan el broche de oro a nuestra creación musical.

ESTUDIOS CASEROS

Otro de los temas que trataremos en esta sección de la revista será la elaboración de estudios de sonido semi-profesionales.
Una vez analizado el software y el hardware utilizado veremos diferentes presupuestos para hacer nuestro propio estudio. Estos presupuestos dependerán de la orientación que queramos dar a nuestra música y de la finalidad profesional de ésta. Por ejemplo, si vamos a componer canciones que incluyan voz, el equipo base puede ser el mismo que si componemos canciones instrumentales, pero en este último caso podemos ahorrarnos algún componente que sólo es útil si se va a grabar voz.

ROCK, TECNO Y OTROS ESTILOS MUSICALES

Uno de los principales objetivos de la banda sonora es hacer una ambientación acorde con el aspecto gráfico y la situación del juego en el tiempo. Aunque en esta sección no vamos a estudiar composición musical, lo que sí haremos será dar los conceptos y las nociones básicas de composición en los diferentes estilos musicales más utilizados en los videojuegos. Los tres estilos más utilizados son el rock, el tecno y la música orquestada.
El primer estilo se utiliza mucho en juegos de acción trepidante, como son los juegos de

carreras de coches, de motos, de lucha, etc. El segundo (el tecno o dance) es tal vez el más extendido, debido fundamentalmente a que en la música sintética muchos de sus componentes se fabrican orientados hacia este estilo en concreto. Este estilo se utiliza sobre todo en juegos con ambientación futurista o en juegos en los que el factor espacio tiempo no está totalmente definido. El tercer y último estilo (la música orquestada) es tal vez el más complejo (ya que conviene tener alguna que otra noción de armonía musical), aunque como veremos más adelante, con un poco de paciencia conseguiremos componer auténticas bandas sonoras al más puro estilo cinematográfico. Este estilo se utiliza para conseguir ambientaciones épicas o situaciones de máxima tensión.

Aparte de estos tres estilos, existen otras corrientes musicales que se utilizan para la música de los videojuegos (New age, por ejemplo). Pero como hemos dicho anteriormente, el rock, tecno y la música orquestada serán el centro de nuestro análisis. Tenemos que tener en cuenta que si conocemos estos tres estilos, podremos hacer cualquier otro tipo de música, ya que siempre será una *fusión* de los tres estilos principales.

COMPARATIVAS

En esta sección haremos comparativas de hardware y software musical. Analizaremos las diferencias y similitudes entre tarjetas de sonido, teclados maestros, secuenciadores y editores de ondas.

Aunque el enfoque de esta sección de la revista sea principalmente el de la música de videojuegos, todo lo que analicemos será útil para cualquier desarrollo musical en el que el ordenador sea *el instrumento base* de nuestras composiciones.

MÚSICA EN EL CD DE PORTADA

En el CD que acompaña la revista incluiremos demos o versiones *shareware* de los programas que analicemos en esta sección. También tendremos canciones compuestas por músicos de videojuegos explicando los diferentes trucos que se deben utilizar para lograr determinadas atmósferas ambientales.

MÚSICA EN INTERNET

Cada artículo de esta sección contendrá diferentes direcciones de internet en las que encontrar información sobre todo lo que tratemos. Para empezar, aquí tenéis una de las direcciones más interesantes en lo que a música midi se refiere: *http://www.midifarm.com/*

"DEMO SCENE"

La aceptación de la informática y los videojuegos en España ha dado lugar a una corriente informática denominada *demo scene*. Programadores y músicos se unen para realizar una demo en la que la música va acompasada al trabajo del programador. En la actualidad, hay reuniones en ferias informáticas en las que los músicos muestran sus canciones compitiendo unos con otros en diferentes modalidades y estilos de composición. Para estos músicos tendremos algún apartado en el que explicaremos y analizaremos el material necesario para hacer este tipo de música. Aquí es donde los formatos MOD, S3M y XM ganan la batalla, así que como veis tendremos material para todos los gustos.

Nuestro primer sonido

A continuación os mostraremos cómo se crea un sonido de guitarra con distorsión.

Una vez instalado el programa en nuestro ordenador, desde Windows 3.X o Windows 95 ejecutamos el archivo cool.exe; en ese momento aparecerá una pantalla en la que nos darán a elegir diferentes opciones. Como es una versión shareware sólo podemos seleccionar dos de las ocho opciones; en nuestro caso, seleccionamos la opción de "*Flanger, Delay and Compressor*" y la de "*Distorsion, Quick filter and Reverse*".

A continuación aparecerá la pantalla principal del programa. Lo primero que haremos será seleccionar la opción "*New*" del "*File Menu*" y aparecerá un cuadro en el que configuraremos la calidad del sonido. Los parámetros que recomendamos son: mono, 16-bits y 44100. Pulsamos OK y ya tenemos todo preparado para crear nuestro primer sonido.

En el *Generate Menú* seleccionamos *Tones*; aparece una ventana con gran cantidad de opciones que se va a configurar como queramos pero, en este caso, vamos a hacer lo siguiente:

"EDITOR DE ONDAS COOL EDIT".

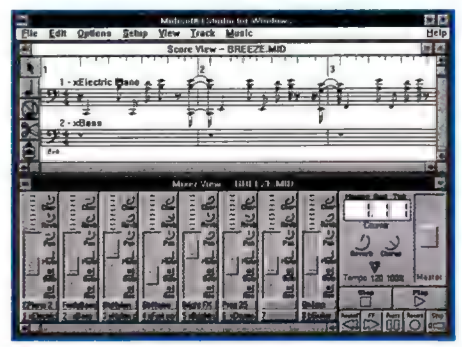
En la parte derecha de la ventana hay cuatro "*Presets*".

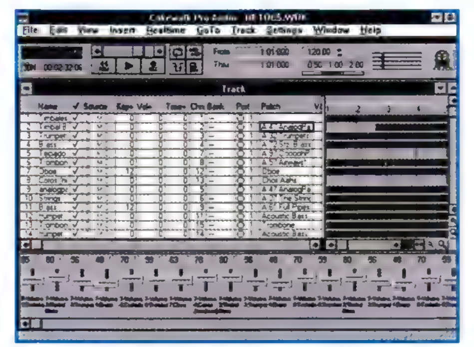
Seleccionamos (con doble click) el *preset* llamado *Chord* (acorde). Como podéis observar, los valores del *Frequency components* han sido modificados automáticamente.

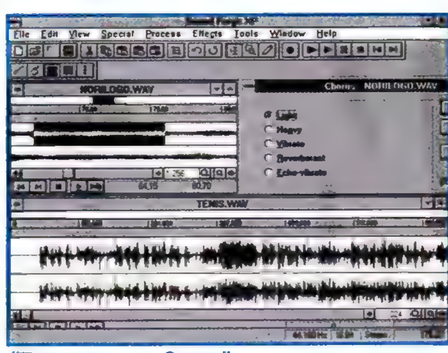
Podemos ir modificando los diferentes valores que aparecen en esta ventana. Aconsejamos que en el valor referente a la duración del sonido no pongáis más de 2 o 3 segundos, ya que si pretendemos hacer sonidos para introducirlos en la memoria de nuestra tarjeta de sonido, cuanto más cortos los hagamos, más espacio libre quedará para introducir más sonidos. Pulsamos OK. En la pantalla principal aparecerá el acorde que hemos generado. Para oírlo basta con pulsar el botón *Play*.

A continuación seleccionamos la opción *Distorsion* del *Transform Menú*, entonces aparecerá otra ventana con diferentes *presets*. Con el ratón hacemos doble click en el *preset* llamado *Overdriven*. Pulsamos OK y vemos cómo cambia la onda de la pantalla principal.

Si pulsamos el botón *Play* escucharemos un sonido de guitarra con *Overdrive*, ideal para utilizar en nuestras composiciones más rockeras.


EDICION DE PARTITURAS A NIVEL PROFESIONAL.


AUDIO Y MIDI EN NUESTRAS COMPOSICIONES.



"TRATAMIENTO DE ONDAS".

Autor: Rafael S. Gago

Cómo crear un escenario para un videojuego

Descubre los secretos que los creadores de videojuegos utilizan en sus trabajos para crear los paisajes y los escenarios que, posteriormente, serán utilizados en algunos de los múltiples tipos de juegos que salen al mercado.

Muchas veces apreciamos cómo un videojuego nos sorprende con paisajes modelados en 3D que pueden llegar a parecer casi reales. Lo mismo los podemos encontrar en escenas estáticas renderizadas, que formando parte de la acción principal del propio juego, esto es, en un escenario *poligonal* o *voxelizado*. Rara vez nos aventuramos en averiguar cuál puede haber sido el método que los diversos grafistas han utilizado, pero si tomamos la decisión de intentar emular el resultado de uno de estos fotorrealísticos paisajes, nos damos cuenta que la cosa tiene que tener algún truco, que no es algo tan sencillo como ponerse delante de un programa de modelación en 3D, hacer una malla con montañas y aplicarle una textura, una luz y una cámara. En este artículo vamos a desvelar uno de los secretos del mundo del desarrollador gráfico.
Para ello, lo primero que tendremos que tener en cuenta es un punto fundamental (más bien podríamos decir que es el punto base de todo el posterior desarrollo), como es la colaboración entre los llamados grafistas o desarrolladores 2D y los grafistas 3D o infografistas; esta colaboración tiene que ser muy estrecha y perfectamente coordinada, pues un buen resultado depende por completo de que ésta esté consolidada desde un primer momento.
Consideraremos asumido que los compañeros que realizan el trabajo sobre papel y sobre software 2D nos ofrecen los resultados que ellos han conseguido y de los cuales, como hemos dicho, depende el resultado final de todo el trabajo. Para ello, podemos usar las imágenes que nos ofrece el artículo de Grafismo 2D.
Para empezar, tenemos que tener una imagen llamada *"mapa de alturas"* que el grafista 2D debe haber llevado a cabo anteriormente con un programa de retoque fotográfico (por ejemplo, Photoshop). Una vez que tengamos la imagen, supongamos que es de un tamaño de 1024x1024 pixeles en 256 tonos de gris o *grayscale*, entraremos en el Vista Pro (es indiferente si se utiliza la versión de Windows o la de DOS).

TODO COMIENZA CON UN "MAPA DE ALTURAS".



PHOTOSHOP DEVUELVE UNA IMAGEN CON SOMBREADO.



CREANDO LA TEXTURA DEL TERRENO

Los siguientes pasos a efectuar consisten en entrar en el menú *Cmap* de Vista Pro, y desde él retocar los colores que darán vida al terreno.
Conviene tener en cuenta que todos los ríos que originemos con Vista Pro van a formar parte de una textura, es decir, que no van a modificar la malla para crearles un lecho, a no ser que éste haya sido previamente ideado por el grafista 2D y preparado en el mapa de grises. Éste es un detalle que puede modificar substancialmente el resultado final, pues no es igual de vistoso un terreno en el que el río al caer de una montaña transcurre por su propio lecho, algo socavado en la tierra, que otro en el que el río en cuestión parezca que se ha pintado sobre la tierra sin ningún tipo de preparación anterior del terreno.
Otro aspecto que conviene tener en cuenta es el que hace referencia a todos aquellos lugares que pretendamos llenar de agua para transformarlos en lagos o mares; si bien la opción de Vista Pro, *Generate Waves*, puede añadir algo de realismo al producir diversidad de colores para simular un movimiento del agua, ésta va ser, en definitiva, parte de una textura, por lo que, si al final del trabajo nos fijamos detenidamente en esta parte del mapa comprobaremos que, de cualquier manera, el agua parecerá que se ha pintado sobre la tierra del fondo y que escala por las orillas (cosa que, en definitiva, es totalmente cierta). Pero esto no va a resultar ningún impedimento, ya que si bien no es conveniente usar un mapa en el que haya gran cantidad de lagos, ríos o mares, tampoco hay que huir de ellos, pues un paisaje siempre queda mucho más vistoso si está adornado con las suaves tonalidades azules del agua.
Como en este artículo no se puede dar una explicación del funcionamiento de Vista Pro, se dará por sentado que el resultado final, en lo tocante a colorido, tonalidad y la forma de llegar a éstos, es conocida por el infografista, de manera que únicamente se darán algunos consejos útiles como guía para la consecución de un mapa satisfactorio.

No es tampoco aconsejable abusar del parámetro *CLIFFS* (riscos), si éste se encarga de producir una tonalidad gris (a ajustar en *CMAP*) para simular las partes de roca viva que quedan en la montaña tras desprenderse partes de ésta. La razón es que tal "desprendimiento" no provocará un alisamiento de la superficie de la roca que quede en la montaña tras desprenderse una parte de ella, sino que conservará su aspecto más o menos redondeado (a no ser que esto ya haya sido previsto, obviamente) y en lugar de añadir realismo puede resultar chocante. En los casos en que la tonalidad de este parámetro se acerque al rojo (como, por ejemplo, si queremos hacer un terreno similar a las montañas del Cañón de Colorado) el resultado, por el contrario, puede ser realmente espectacular.

Como se ha dicho, se reconocerá que el aspecto del mapa en Vista Pro es satisfactorio, de manera que una vez que nos guste el resultado deberemos salvar la textura. Para ello, lo primero que haremos será abrir el menú *GrMode* y activar la opción *Enable 24 Bit*, tras lo cual, desde el menú *Save*, elegiremos *Save Texture Map* y conservaremos la textura que, si bien nos va a parecer simple y sin ninguna posibilidad de llegar a ningún lugar decente, tras pasar por las manos de los grafistas 2D y por los filtros de PhotoShop parecerá algo totalmente diferente a lo que ha salido de Vista Pro. Con esta nueva textura, que ahora tenemos gracias al proceso de retoque en 2D, y con la ya conocida imagen de alturas con la que empezamos el trabajo en Vista Pro, comenzaremos la segunda parte del proceso de elaboración del terreno, para lo cual cambiaremos de software y entraremos en el mundo del 3D-Studio4.

Pasos a seguir para la creación de un escenario 3D

1º.- Partir de un mapa de alturas en escala de grises, con el que comenzaremos la creación a partir del software VistaPro.

2º.- Transformaremos el mapa en un archivo PCX de 256 tonos de gris (indexado) de un tamaño recomendado de 1026 x 1026 píxeles.

3º.- Desde VistaPro escogeremos la opción PCX->DEM en el menú ImpExp y tomaremos la imagen de grises que desde ahora denominaremos "Mapa de Alturas".

4º.- Desde el menú Cmap de VistaPro, daremos al mapa de alturas los colores que resulten más apropiados al tipo de orografía ideada.

5º.- Una vez concretados los tonos del recién creado escenario tridimensional, tomaremos la opción Save Texture Map del menú *SAVE* teniendo presente que, para ello, antes debemos activar la opción Enable 24 Bit del menú *GRMode*. Recordemos que la imagen a salvar no será sino un cúmulo de colores sin demasiada definición que cumplirá por completo su efecto cuando sea sombreada por los grafistas 2D desde el PhotoShop.

6º.- Con la imagen retocada y sombreada del terreno, deberemos abrir el 3D Studio y usar la opción GRID del submenú *PXP Loader* bajo la barra Program, usando los valores de 100 en las casillas *LENGHT* y *WIDTH* y 0 en la casilla *HEIGHT*, y con un número grids de 100 en las correspondientes a los de los primeros y 0 en la del último.

7º.- En el mismo menú usaremos la opción *DISPLACE* para levantar el terreno partiendo del anterior "mapa de alturas". Un valor de 20 en la casilla *STRENGH* será suficiente para dar un relieve apropiado. Debemos recordar que la imagen de mapa de alturas que debemos usar para efectuar el desplazamiento ha de estar en un formato que reconozca el 3D Studio, por lo que no es válido el PCX que hemos usado para el VistaPro.

8º.- Desde el editor de materiales le asignaremos el proporcionado tras el retoque de PhotoShop, cuidando que la iluminación del 3D Studio tenga la misma dirección que la que se le ha dado anteriormente en PhotoShop.

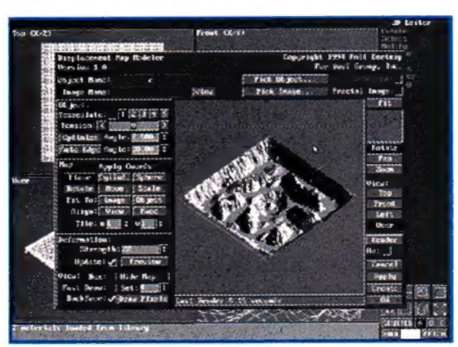
NOTA ESPECIAL

Si, en lugar de usar el 3D Studio 4, queremos usar el 3D Studio MAX deberemos crear una malla de tipo *CUAD PATCH* con unos valores de 200 para ancho y alto y un número de 5 en ambas casillas de segmentos. En el Menú de Modificadores le añadiremos el paso *EDIT PATCH* y le marcaremos 10 en la casilla *TOPOLOGY STEPS*, tras lo cual se transformará en malla editable con la opción *EDIT MESH*. Ahora estamos preparados para aplicar la opción *DISPLACE* y seguir los pasos de forma similar a los llevados a cabo con el 3D Studio 4

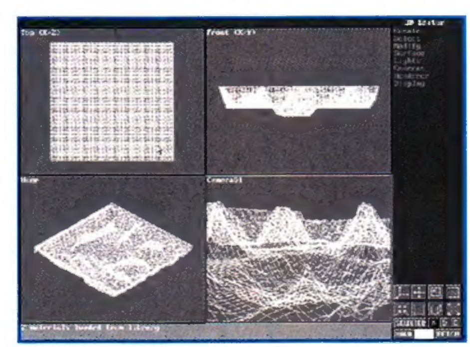
LEVANTANDO EL TERRENO

El primer paso a realizar con el 3D Studio es el de crear una red con la opción *GRID* desde el menú *PXP LOADER*, con unas dimensiones de 100 en las casillas *LENGHT* y *WIDTH* y 0 en la casilla *HEIGHT*, y con un número *grids* de 100 en las correspondientes a los dos primeros y 0 en la del último. Tales parámetros pueden ser modificados a gusto del infografista y en dependencia, una vez más, de la potencia de la máquina usada. Tras ello, regresaremos al menú *PXP LOADER* y tomaremos la opción *DISPLACE*. En el cuadro de imagen usaremos el primer mapa de alturas en gris que utilizamos para Vista Pro, pero hay que recordar que si bien Vista Pro necesitaba una imagen en formato PCX, el 3D Studio no soporta dicho formato, por lo que habrá que convertirlo en cualquier otro que pueda usar, por ejemplo, *GIF*. En la casilla *STRENGTH* tomaremos el valor que deseemos para que la elevación de la orografía sea satisfactoria; un valor de 20 puede resultar apropiado.

El siguiente paso es abrir el editor de materiales y tomar la imagen con textura de Vista Pro que, posteriormente, ha sido retocada y sombreada con PhotoShop. Después, asignaremos la textura a nuestra malla recién creada con relieve y le aplicaremos el mapa de coordenadas de tipo plano. Ahora podemos añadir una luz y realizar el primer render para comprobar como está quedando la textura aplicada. Si el resultado es satisfactorio pero queremos conseguir un poco más de realismo, podemos usar un mapa de volumen (*bump*) en la casilla correspondiente del editor de materiales (un valor de 10 resulta suficiente). Una vez hecho esto, el proceso final será el de recolocar las luces y la cámara de forma que quede resaltado el relieve del mapa, por lo que se puede añadir una imagen de fondo (*background*), y algo de niebla de distancia (*fog*) en el supuesto de querer realizar un render. En caso de que todo el trabajo haya sido enfocado para ser utilizado, posteriormente, en el motor del juego (un escenario poligonal) el trabajo que resta corresponde a los programadores.



3D-Studio creamos un mapa de desplazamiento.



Con la malla creada, el trabajo esta casi completo.



La imagen puede usarse en un videojuego.

Autor: Carlos Cabañas

Texturas de terrenos

Puede parecer que en un juego tanto el grafismo 2D como el 3D están separados, teniendo cada uno su propio campo y que, si bien se unen para dar un resultado final, durante el camino van paralelos.

Pero como vamos a comprobar muy pronto, la unión del trabajo de un grafista 2D y de un infografista tiene que ir pareja para que el resultado sea el deseado.

Para mostrar la idea de la conjunción existente entre ambos tipos de trabajo, se puede utilizar una imagen que, aunque parezca sencilla, se comprobará en el transcurso de la realización que resultaría imposible llegar a su finalización sin la perfecta sincronización de los dos caminos. La imagen en cuestión no es otra que una nave espacial sobrevolando un terreno.

El punto culminante de dicha imagen es el terreno, en el que se va a volcar todo el trabajo, tanto de los grafistas 2D como de los de 3D y el software a utilizar será el siguiente:

- Software desarrollo 2D:
  Adobe PhotoShop 3.0 (o 4.0).
  Fractal Design Painter 2.5 (o 4.0).
- Software desarrollo 3D:
  Vista Pro 3.0 versión para MS-D.O.S. (también es factible el uso de la versión para Windows).
  3D-Studio versión4 (también es factible el uso de 3D Max).

Para comenzar la nave hay que realizar, a mano alzada en un papel, el boceto del terreno que pretendemos usar en la imagen. Con él tendremos una idea clara de las irregularidades del suelo, los montes, las llanuras y de toda la orografía que vamos a plasmar en lo que será el mapa sobre el que vayamos a colocar la nave. Una vez hecho esto, deberemos hacer una composición del aspecto del mapa desde el punto de vista de la altura que vaya a tener, esto es, cómo se verá el mapa desde arriba, pues así es como se entregará la imagen al infografista para que realice el trabajo en tres dimensiones del mapa. De la misma manera, esta imagen primera va a ser la que use en varias ocasiones el propio desarrollador 2D para dar el aspecto final a los colores del terreno.

Una vez se tenga claro todo esto, se procede a plasmar en un programa de diseño 2D lo que se ha ideado sobre el papel. Para ello haremos lo siguiente: el primer paso consiste en escanear el boceto a mano del mapa visto desde arriba para proyectarlo sobre el programa de diseño 2D (en este caso el PhotoShop).

En el caso de no poder hacer uso de un escáner, con lo que conseguiríamos un ahorro importante de tiempo y de esfuerzo, habrá que hacer a mano con PhotoShop una imagen lo más fiel posible a lo que plasma el papel (hay que tener en cuenta que se va a usar una referencia muy precisa a las alturas y a los accidentes del terreno). Habrá que tener en cuenta, desde un primer momento, el tamaño de la imagen ya que el Vista Pro tiene unos tamaños predefinidos, por lo que se usará una imagen de 1024 x1024; la razón para usar este tipo de imagen es que, si bien lo ideal sería hacerla del máximo tamaño que permite Vista Pro, esto es 2050 x 2050, para que la definición de la textura que éste exporta sea idónea, un tamaño de imagen de estas características requeriría una máquina con unas prestaciones un tanto elevadas y, al reducir el tamaño, se pueden conseguir resultados igualmente satisfactorios con un ahorro en tiempo de proceso.

Una vez escaneada la imagen procederemos a modificar su tamaño al anteriormente indicado. Tras ello, convertiremos la imagen a escala de grises teniendo en cuenta que el blanco puro

IMAGEN 1.



IMAGEN 2.





IMAGEN 3.



IMAGEN 4.

(255 en la escala de color) será representación de la altura máxima y el negro (0 en la escala de color) será la representación de la mínima altura. Para tener versatilidad de las alturas utilizaremos un tono de gris más o menos intermedio como nivel para nuestro suelo (70 en la escala de color).
La razón por la que no utilizamos el valor realmente intermedio, que sería 127, viene dada porque siempre utilizaremos más tonalidades de gris para dar altura a las montañas que las que usaremos para preparar el lecho de los ríos o valles bajos o cualquier accidente geográfico por debajo del nivel del suelo.
Utilizaremos la herramienta de Aerógrafo con una presión de un 100% y un tamaño ajustado al trazado a realizar; una Pluma no demasiado difuminada ya que después se usará el difuminar propiamente dicho para crear las irregularidades que dotarán de mayor realismo a la escena. Con el Aerógrafo pintaremos completamente de blanco las cimas de las montañas, con el gris intermedio los lugares que correspondan al "nivel del suelo" y con el negro los lugares que representen los niveles más bajos de la escena (Imagen 1). Una vez terminada la representación base de las alturas pasaremos a utilizar el difuminado.(difumino)
Lo que logramos con éste es que la diferencia entre blancos y negros quede menos brusca con lo que se evitarán riscos escarpados que a la hora de proyectar la imagen en 3D resultarían demasiado llamativos y podrían llegar a romper la armonía de la escena. Con esta herramienta se dotará al terreno de la morfología precisa y sensación de realidad, porque el resultado tendría que quedar lo más similar a una fotografía aérea (Imagen 2).
Con esta técnica no se pueden realizar ni túneles ni puentes, sino que hay que crearlos de manera externa e insertarlos en el terreno como objetos 3D.
Una vez terminada la representación de las alturas salvaremos la imagen de dos maneras diferentes: una con extensión PCX, que se la daremos al infografista para que la trate con el VISTAPRO, y que, posteriormente, nos devuelva la textura en color (este apartado se explica en el Taller 3D de la revista) para modificarla y dejar el resultado definitivo para exportarla al 3d_MAX; la otra, por ejemplo, con extensión Gif pues la necesitaremos para el MAX ya que éste no trabaja con ficheros con extensión PCX.
Una vez tomada la imagen del Vistapro (Imagen 3), se abrirá la imagen en escala de grises (*.PCX) que estaba salvada. Lo primero que hay que hacer es asegurarnos que ambas imágenes tienen exactamente el mismo tamaño. Una vez hecha la comprobación pasaremos a realizar la elevación del terreno en la imagen RGB. Para esto seleccionamos la imagen en escala de grises, una vez en ella abrir la ventana de canales (*Menú ventana, mostrar canales*) y seleccionar todo (*Menú selección, todo*) en el canal negro que nos encontramos y copiamos (*Menú editar, copia*); terminado esto podemos cerrar la imagen.
En la imagen, en RGB, seleccionamos la ventana de canales y creamos un nuevo canal (*#4*) y lo seleccionamos; una vez en él pegamos (*Menú editar, pegar*) el canal negro de la imagen de escala de grises. Activamos el canal RGB, desactivando el canal *#4*, y pasamos a la ventana de capas (*Menú ventana, mostrar capas*) que es donde se centrará todo el trabajo.
Para dar la sensación de 3D aplicamos el filtro efecto de luces (*Menú filtros, render, efectos de luces*). En la ventana que aparece seleccionamos una luz direccional para que la escena reciba por igual la luz, ya que luego el infografista dispone de una mayor flexibilidad para modificar la luz general de la escena en 3D.
Seguidamente, en el submenú canal de textura, seleccionamos el canal *#4* (escala de grises) y activamos el botón blanco arriba (observaremos que nuestra textura parece tomar volumen); la variación de la altura se la aplicaremos dependiendo de la cantidad de contraste que deseemos para la imagen.
Salvamos la imagen, cerramos Photoshop y abrimos el *Fractal Design Painter*. Con este programa dotamos a nuestra textura de una sensación auténtica de terreno a través de una serie de filtros avanzados de modificación de superficies que dispone. Vamos a utilizar este programa ya que está más desarrollado que Photoshop, pero no es muy necesario ya que con Photoshop, con su filtro texturear (*Menú filtros, textura, texturizador*), se pueden lograr resultados aceptables, aplicando, por ejemplo, el mapa de relieve *sand*, con un grado de relieve aproximado de 2.
Una vez terminado esto sólo resta dar algunos pequeños retoques como, por ejemplo, marcar surcos en los caminos, crear olas en el agua, etc.
Una vez terminada (Imagen 4) se la pasaremos al infografista para que la aplique en el modelo 3D del terreno, con un resultado final bastante aparente.

El Emisario

Steven Jobs vuelve a casa



Steven Jobs, el padre del Macintosh y fundador de Apple Computer, después de "abandonar" la presidencia de la compañía a los 30 años y tras pasar 12 años inmerso en diversos proyectos (entre ellos, la fundación de la compañía NeXT y la creación del estudio Pixar, responsable de la película Toy Story), vuelve a Apple para sacarla de la apurada situación que está pasando últimamente. Mediante la compra de NeXT por parte de Apple, Steven se convierte en el nuevo asesor de "la manzana", pero lo que verdaderamente ha asombrado a todo el mundo ha sido el acuerdo firmado entre Jobs y su máximo enemigo Bill Gates para apoyar la caída económica de Apple.

Maximun Power Memory

Intel ha mostrado al mundo su nuevo chip de memoria que cambiará radicalmente el uso del ordenador. La tecnología desarrollada por los ingenieros del gigante de los microprocesarodes modifica la estructura de los chips permitiendo guardar los datos incluso con el ordenador apagado. El logro de almacenar dos bits donde hasta el momento solo se podía almacenar uno, posibilita que la **StrataFlash** albergue 64 Mb en un sólo chip. De momento, esta tecnología sólo se aplica en las tarjetas PCMCIA y en los ordenadores de bolsillo.

Máximo apoyo de Microsoft a Windows NT

Durante la Conferencia profesional de desarrolladores celebrada en San Diego, Jim Allchin, vicepresidente de Microsoft, anunció que después de la aparición de Windows 98, que se estima aparecerá en el segundo cuarto de 1998, Microsoft centrará sus fuerzas en Windows NT, descartando el desarrollo de cualquier otra versión de Windows noventai...

Revisión de DirectPlay

Los usuarios de DirectX 5.0 deberían bajarse esta pequeña (227 Kbs) revisión de DirectPlay, que soluciona pequeños fallos que posee la versión original. Para instalarlo, ejecuta el archivo PLAY50A.EXE una vez te lo hayas bajado de http://www.microsoft.com/directx/resources/downloads/dplay50a.exe.

LOS MEJORES Y MÁS BUENOS

Los creadores de *Command&Conquer*, Westwood Studios, en una muestra de gratitud y buen hacer, y velando por la educación de los más jóvenes dentro del mundo de la informática han donado diverso material informático, valorado en unos 140.000 dólares, a la Universidad de las Vegas. Ordenadores que van a ser destinados a los departamentos de arte, arquitectura y diseño gráfico. A parte de los equipos, Westwood ha entregado, en un alarde de inmensa gratitud, software de diseño por valor de 200.000 dólares.

Virgin se marca un tanto

Con el lanzamiento de NHL Powerplay'98, para Sony Playstation y PC CD-Rom para Windows 95, la tan esperada segunda parte, proporciona el desafío y el realismo del hockey profesional con inteligencia artificial avanzada e increíbles gráficos en 3D.

Avisamos

El primer semestre del próximo año será decisivo para el futuro del software de entretenimiento desarrollado en España. STOP. Os mantendremos informados. STOP.

Lara Croft se enrolla con Sony

La compañía Sony Entertainment y Eidos Interactive han firmado un contrato con el que obtienen los derechos en exclusiva de las secuelas, versiones, ampliaciones y demás, del exitazo superventas Tomb Raider. Por lo que a partir de ahora sóo veremos a Lara Croft en la consola Play Station de Sony. Pero que no cunda el pánico entre la gente del PC pues, el contrato, sólo incumbe al terreno consolero, tenemos Lara para rato.

Si quieres colaborar con nosotros en la sección "El Emisario" envíanos tus noticias, ocurrencias, ... y todo lo que se te pase por la cabeza a la siguiente dirección.

EL EMISARIO
Alfonso Gómez 42. Nave 1-1-2
28037 Madrid

El Emisario educa y divierte

